На предыдущем занятии мы с вами начали создавать класс DArray для реализации идеи динамического массива. И остановились на необходимости переопределения операции индексирования. Первое, что приходит в голову, это объявить в классе соответствующий метод:

```language-cpp
operator[](int index) { }
```

Но здесь сразу возникает вопрос, что должен возвращать этот метод? Если просто возвратить целочисленное значение:

```language-cpp
    int operator[](int index)
    {
        return data[index];
    }
```

то получим, при использовании объекта этого класса:

```language-cpp
ar1[5] = 7;   // ошибка
int v = ar1[3]; // ok
```

У нас выражение `ar1[5]` не является леводопустимым (l-value), то есть, не связано с ячейкой памяти, в которую можно заносить какое-либо значение. Правда, если при определении оператора указать ссылку:

```language-cpp
    int& operator[](int index)
    {
        return data[index];
    }
```

то компилятор не выдаст никаких ошибок. Но, я думаю, большинство из вас понимают, что такая реализация операции индексирования нас не может устроить. Здесь есть очевидные проблемы. **Во-первых, если индекс будет указан за пределами массива data, то запись или чтение будет происходить с неопределенными ячейками памяти, что недопустимо**. Но это, конечно, поправить не сложно, прописав условие для переменной index:

```language-cpp
    int& operator[](int index)
    {
        if(index >=0 && index < capacity)
            return data[index];
```

```
        return ???;
    }
```

Правда, нужно еще подумать, что возвращать, если индекс выходит за пределы массива.

Вторая проблема посерьезнее. Если выполняется операция присваивания по определенному индексу:

```language-cpp
ar1[15] = 7;
```

и этот **индекс превышает физический размер массива data, то по идее динамического массива, его следует увеличить до нужного размера** и занести в нужный элемент соответствующее значение. **А если выполняется чтение**:

```language-cpp
int v = ar1[15];
```

то никакого изменения массива data выполнять не нужно, даже если индекс превышает физический размер.

Спрашивается, как нам в программе различать эти два варианта использования операции индексирования? К сожалению, простого решения здесь нет. Поэтому нам придется сделать «хитрый» прием.

В классе DArray объявим еще один вспомогательный вложенный класс Item. Объект класса Item будет возвращаться операцией индексирования. А в самом классе мы переопределим две операции:

- операцию присваивания;
- операцию преобразования класса к типу int.

Операция присваивания будет срабатывать в момент присвоения массиву значения, а операция преобразования типа – в момент чтения значения. Так мы сможем различить эти две ситуации.

Давайте реализуем эту логику. Первым делом в классе DArray объявим вложенный класс Item:

```language-cpp
class DArray {
    enum {
        start_length_array = 8, // начальная длина массива
        resize_factor = 2,      // множитель для увеличения длины массива
        max_length_array = 30,  // максимальная длина массива
```

```
        value_error = 2123456789, // специальное значение для обозначения ошибки данных
    };

    class Item {
        DArray* current {nullptr};
        int index {-1};

    public:
        Item(DArray* obj, int idx) : current(obj), index(idx)
            { }

        operator int() const;
        int operator=(int right) const;
    };

    int* data = nullptr;
    int length {0};   // число записанных в массив значений
    int capacity {0}; // физический размер массива

    void _resize_array(int size_new); // увеличение размера массива data но не более max_length_array элементов
public:
...
};
```

В классе Item объявлен указатель current на объект класса DArray – динамического массива, с которым выполняется работа, и целочисленная переменная index – индекс элемента, к которому идет обращение через операцию индексирования. Эти переменные инициализируются в конструкторе с двумя параметрами, то есть, создать объект класса Item без аргументов не получится, нужно обязательно передать текущий объект класса DArray и индекс элемента. Далее, идут объявления двух методов, о которых мы только что говорили. Обратите внимание, что у операции преобразования типа возвращаемый тип прописывается после ключевого слова operator.

В самом классе DArray операция индексирования примет вид:

```cpp
Item operator[](int index)
{
return Item(this, index);
}
```

Мы здесь возвращаем копию объекта, что нас вполне устраивает. Осталось записать реализации методов вложенного приватного класса Item в файле darray.cpp. Они будут следующими:

```cpp
DArray::Item::operator int() const
{
    if(index >= current->length || index < 0)
        return value_error;

    return current->data[index];
}
```

```cpp
int DArray::Item::operator=(int right)
{
    if(index >= max_length_array || index < 0)
        return right; // размер массива data не может превышать max_length_array элементов

    if(index >= current->capacity) {
        current->_resize_array(index+1);
    }

    for(int i = current->length; i < index; ++i)
        current->data[i] = 0;   // зануляем все новые добавленные значения

    if(index >= current->length)
        current->length = index + 1; // новый размер записанных данных

    current->data[index] = right;
    return right;
}
```

language-cpp

В методе операции преобразования типа мы вначале проверяем корректность индекса. Если он выходит за пределы данных, занесенных в массив, то возвращается специальная предопределенная константа value_error. Это сделано для удобства в учебном проекте, чтобы не усложнять программу.

В методе операции присвоения так же вначале идет проверка на допустимое значение индекса. Если оно выходит за установленные пределы, то ничего не присваивается, а просто возвращается присваиваемое значение. Далее,

проверяется, если индекс превышает физический размер массива data, то вызываем метод класса DArray для увеличения массива до нужных размеров. **Обратите внимание, что в методах вложенного класса Item можно совершенно спокойно обращаться к приватным элементам внешнего класса DArray**. После увеличения массива (если это было необходимо), зануляются все промежуточные элементы, а в элемент с индексом index заносится присваиваемое значение right. В конце так же возвращается эта величина.

Давайте посмотрим, как будет работать операция индексирования:

```language-cpp
int main()
{
    DArray ar1, ar2;

    for(int i = 0;i < 10; ++i)
        ar1.push_back(i+1);

    ar1[14] = 7;
    int v = ar1[14];

    ar2 = ar1;

    std::cout << v << std::endl;
    std::cout << ar2.size() << " " << ar2.capacity_ar() << std::endl;

    for(int i = 0;i < ar2.size(); ++i)
        std::cout << ar2.get_data()[i] << " ";

    return 0;
}
```

После запуска программы в консоли увидим:

```
7
15 16
1 2 3 4 5 6 7 8 9 10 0 0 0 0 7
```

Как видите, все отработало успешно. Причем, операция приведения типа, которую мы записали, как int, будет так же срабатывать и при присвоении значения переменной любого другого числового типа. Например:

```language-cpp
double v = ar1[14];
```

Компилятор автоматически выберет подходящее преобразование объекта класса и применит его. **Если автоматическое скрытое преобразование типа объекта не допустимо, то мы можем воспользоваться знакомым ключевым словом explicit** следующим образом:

```language-cpp
explicit operator int() const;
```

Тогда в коде программы придется прописывать конструкцию вида:

```language-cpp
double v = static_cast<int>(ar1[14]);
```

О преобразовании типов классов (функции static_cast и некоторых других) мы еще будем говорить.

Из этого занятия вы должны хорошо себе представлять, как переопределяется операция индексирования и какие особенности ее работы существуют. Как записывается операция преобразования типа класса. А также, как объявляются и работают вложенные классы.